

Tool for Behavioral Analysis of Well-Structured Transition Systems

Leonid Dworzanski

Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
leo@mathtech.ru

Vladimir Mikhaylov

Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
vemikhaylov@edu.hse.ru

Abstract— Well structured transition systems (WSTS) became a well known tool in the study of concurrency systems for proving decidability of properties based on coverability and boundedness. Each year brings new formalisms proven to be WSTS systems [11,12]. Despite the large body of theoretical work on the WSTS theory, there has been a notable gap of empirical research of well-structured transition systems. In this paper, the tool for behavioural analysis of such systems is presented. The WSTS systems are described via the extension of SETL language. It makes the description of the formalism close to the formal definition. It allows to easily introduce new formalisms and conduct analysis of the behavioural properties without programming efforts. Two most studied algorithms for analysis of well-structured transition systems behavior (backward reachability and the Finite Reachability Tree analyses) have been implemented; and, their performance was measured through the runs on such models as Petri Nets and Lossy Channel Systems. The developed tool can be useful for analysis of different subclasses of well-structured transition systems.

Keywords—formal verification; infinite systems; well-quasi-ordering; Petri net

I. INTRODUCTION

Formal verification provides researchers and developers with approaches that are widely-used for proving that a program satisfies a formal specification of its behavior. These methods are highly demanded in the software and hardware engineering, as they provide appropriate level of systems reliability; which, in most cases, cannot be ensured by simulation.

One of the most common technique of formal verification is model checking or property checking. It involves algorithmic methods that are applied to check satisfiability of a logic formula used for the representation of the model of the system and the specification. The main advantage of model checking is considered to be the fact that it enables almost completely automatic process of verification. Model checking proved to be effective in practice for analysis of finite-state systems [1]; however, in case of systems with infinite state space the situation is more complicated because exhaustive search, which is usually used by verification tools, cannot be applied directly.

In order to deal with infinite-state systems Finkel proposed the idea of well-structured transition systems (WSTS) in 1987 [2]. “These are transition systems where the existence of a well-

quasi-ordering over the infinite set of states ensures the termination of several algorithmic methods. [3]” The suggested model has provided researchers with an abstract generalization of several models (i.e. Petri nets, lossy channel systems and timed automata). Therefore, the results obtained from the analysis of such a generalized model can be also applied to these specific models.

The WSTS analysis can be used to solve, for instance, covering, termination, inevitability and boundedness problems. However, the application of the WSTS analysis is hampered by the necessity of implementing algorithms and data structures to support the analysis for each new formalism. In this work, the tool that can be used for analysis of WSTS is presented. We introduce the WSTSL language - modification of SETL language – set-theoretical programming language. The language provides the user with opportunity to define the structure of analyzed system very close to the original formal definition. After definition of the formalism, it is immediately possible to run backward reachability method [4] or the Finite Reachability Tree [5] on it. It allows a computer scientist to conduct the analysis of a WSTS system almost instantly after proving the well-structuredness of his or her formalism, and to postpone the implementation phase after what-if experiments were conducted successfully.

The rest of the paper is organized as follows. The second section describes WSTS’s basic terms and underlying concepts. The third section provides the description of two used algorithms (the backward reachability method and the Finite Reachability Tree). The fourth section presents the architecture of the developed analysis tool. The fifth section shows how the developed tool is used for the analysis of Petri nets and provides performance analysis results. The sixth section summarizes and provides possible applications of the study for the future research.

II. WELL-STRUCTURED TRANSITION SYSTEMS

The definition of well-structured transition systems (WSTS) was proposed by Finkel in [2]. It is based on the two main concepts: transition systems (TS) and well-quasi-orderings on the states of these systems.

Transition system (TS) is one of the most general and widely-used models for formal description of the behavior of different systems. A *transition system* is defined by a structure

$TS = (S, \rightarrow)$ where $S = \{s, t, \dots\}$ is a set of *states*, and $\rightarrow \subseteq S \times S$ is any set of *transactions* [3]. TS can be also supplemented by initial states, labels for transitions, durations or causal independence relations, and other information [3]; however, for the consideration of the concept of WSTS using of set of states along with labeled transactions is sufficient.

A binary relation \leq on a set X is called *preorder* or *quasi-ordering* (qo) if it is reflexive and transitive. So for any $a, b, c \subseteq X$ we have:

- 1) $a \leq a$ (reflexivity);
- 2) if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).

Definition 1. A *well-quasi-ordering* (wqo) is a quasi ordering in which for every infinite sequence of elements $x_0, x_1, x_2, x_3, \dots \subseteq X$ there exist such indices $i < j$ that $x_i \leq x_j$ [3, 6]. According to [7], there are several equivalent definitions of wqo ; however, the definition given here is generally used in the WSTS theory.

Definition 2. A *well-structured transition system* (WSTS) is a transition system $TS = (S, \rightarrow, \leq)$ equipped with a $qo \leq \subseteq S \times S$ between states such that the two following conditions hold:

- 1) well-quasi-ordering: \leq is a wqo , and
- 2) compatibility: \leq is (upward) compatible with \rightarrow , i.e. for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists such a sequence of transitions $t_1 \rightarrow^* t_2$ that $s_2 \leq t_2$ [3].

$Succ(s)$ denotes the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s . Likewise, $Pred(s)$ denotes the set $\{s' \in S \mid s' \rightarrow s\}$ of immediate predecessors.

An upward-closed set is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. A basis of an upward-closed I is a set I^b such that $I = \bigcup_{x \in I^b} \uparrow x$, where $\uparrow x =^{def} \{y \mid y \geq x\}$.

III. ALGORITHMS

A. Backward Reachability Method

Backward reachability method proposed by Abulla et al. in [4] is intended to solve the covering problem which is to decide, given two states s and t , whether starting from s it is possible to reach a state $t' \geq t$. This is essentially one of set-saturation methods termination of which relies on the lemma that says that any increasing sequence of upward-closed sets ($I_0 \subseteq I_1 \subseteq I_2 \subseteq \dots$) eventually stabilizes (i.e. there is such a $k \in \mathbb{N}$ that $I_k = I_{k+1} = I_{k+2} = \dots$) [3].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$ and some upward-closed set of states $I \subseteq S$. Backward reachability method on the each j -th step generates the set of states from which I can be reached by a sequence at most j transitions [4].

More strict generalization was suggested by Finkel and Schnoebelen in [3], where it involves computing $Pred^*(I)$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq \dots$ where $I_0 =^{def} I$ and $I_{n+1} =^{def} I_n \cup Pred(I_n)$.

Definition 3. A WSTS has effective pred-basis if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow Pred(\uparrow s)$.

The covering problem is decidable for WSTS if it has effective pred-basis and decidable \leq . The proof of this statement is given in [3]. Essentially, it is said that if there is a sequence $K_0, K_1 \dots$ with $K_0 =^{def} I^b$ (finite basis of I), $K_{n+1} =^{def} K_n \cup pb(K_n)$ and m is the first index such that $\uparrow K_m = \uparrow K_{m+1}$, then $\uparrow \cup K_i = Pred^*(I)$. By decidability of \leq , it is possible to check whether $s \in \uparrow Pred^*(\uparrow t)$.

B. Finite Reachability Tree

The Finite Reachability Tree belongs to tree-saturation methods which represent methods that consider all possible computations inside a finite tree-like structure [3]. It is also called the forward analysis method, in contrast to the backward analysis. Essentially, it is based on the ideas proposed by Karp and Miller in [5].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$. For any state $s \in S$, the Finite Reachability Tree is such a finite directed graph (tree) that:

- 1) nodes of the tree are labeled by states of S ;
- 2) nodes are either dead or live;
- 3) the root node is a live node n_0 , labeled by s (written $n_0 : s$);
- 4) dead nodes have no child nodes;
- 5) a live node $n : t$ has one child $n' : t'$ for each successor $t' \in Succ(t)$;
- 6) if along the path from the root $n_0 : s$ to some node $n' : t'$ there exists a node $n : t$ ($n \neq n'$) such that $t \leq t'$, we say that n subsumes n' , and then n' is a dead node [3, 6].

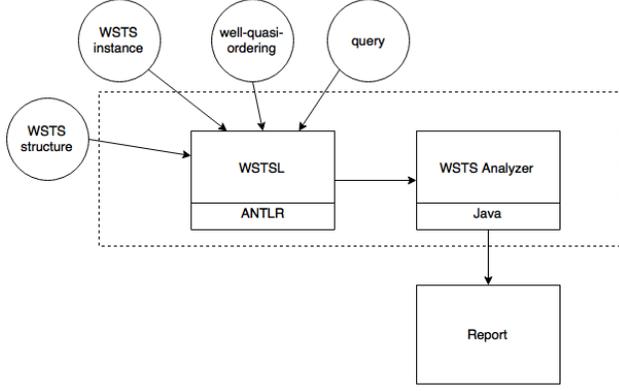
The Finite Reachability Tree is effectively computable if S has (1) a decidable \leq , and (2) $Succ$ mapping is computable [3]. All paths in the finite reachability tree are finite as any infinite path would include a covering node [6].

This algorithm can be applied to termination, inevitability, and boundedness problems (see [3] for details).

IV. PROPOSED ARCHITECTURE

The general structure of the architecture of the developed tool is illustrated in Fig. 1. It consists of two main parts: Well-Structured Transition Systems Language (WSTSL) and WSTS Analyzer. Also there are four input parameters that are set by the user through WSTSL.

Fig. 1. Architecture of the developed tool



WSTSL is a programming language used in the developed system as the front-end which provides user with a means of describing the structure of WSTS and its essential operations and relations: *Succ*, *Pred*, preorder \leq . The supported data types are: integers, tuples, maps and sets. The analysis algorithms are implemented as the commands: *backwardanalysis()* and *forwardanalysis()*. As it is depicted in the Fig.1 the parser for WSTSL is built with the compiler-compiler Another Tool for Language Recognition (ANTLR) [8]. The WSTSL parser's sources are generated in Java.

WSTS Analyzer represents that part of the system which is responsible for the analysis of the transition system, which it gets from the WSTSL parser. WSTS Analyzer is implemented in Java, as it allows to naturally interact with the parser Java classes generated by ANTLR.

As it was noted above, the input that is provided by the user, consists of the four parts. Firstly, a general structure (WSTS structure) of the analyzed transition system should be described. Secondly, a well-quasi-ordering compatible with the defined structure should be specified. Then, a structure of a specific transition system (WSTS instance) that corresponds to the general structure is provided. Finally, the desired analysis algorithm with appropriate parameters (query) is invoked. Essentially, all these parts are described in a single input program written in WSTSL. Afterwards, the WSTS Analyzer runs the selected algorithm on the specified system and generates report which format depends on the choice of the algorithm.

V. EXPERIMENT

A. Petri Net

The applicability of the proposed approach could be demonstrated by an example of a Petri net which is well-structured transition system. The classical definition of this model is the following.

Definition 4. A Petri net (P/T-net) is a 4-tuple (P, T, F, W) where

- P and T are disjoint finite sets of places and transitions, respectively;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs;

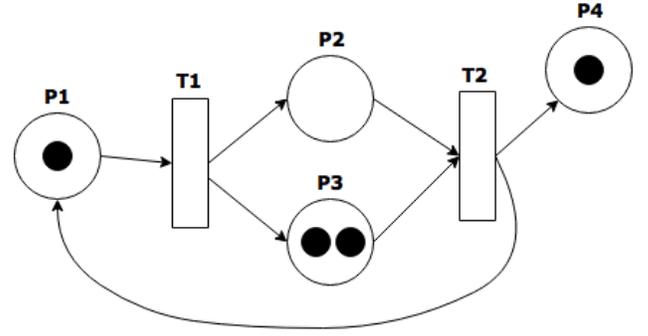
- $W : F \rightarrow \mathbb{N} \setminus \{0\}$ – an arc multiplicity function, that is, a function which assigns every arc a positive integer called an arc multiplicity or weight.

A marking of a Petri net (P, T, F, W) is a multiset over P , i.e. a mapping $M : P \rightarrow \mathbb{N}$. By $M(N)$ we denote the set of all markings of the P/T-net N .

We say that a transition t in the P/T-net $N = (P, T, F, W)$ is active in marking M if for every $p \in \{p \mid (p, t) \in F\}$: $M(p) \geq W(p, t)$. An active transition may fire, resulting in a marking M' , such as for all $p \in P$: $M'(p) = M(p) - W(p, t)$ if $p \in \{p \mid (p, t) \in F\}$, $M'(p) = M(p) - W(p, t) + W(t, p)$ if $p \in \{p \mid (t, p) \in F\}$ and $M'(p) = M(p)$ otherwise.

For the experiment, we will use the Petri net illustrated in Fig. 2.

Fig. 2. A simple unbounded Petri net



First of all, the general structure of the Petri net model described above should be defined by means of WSTSL (Fig. 3).

Fig. 3. General structure of Petri net in WSTSL

```

type P : set of int;
type T : set of int;
type PT(P1:P, T1:T) : set of [from P1, from T1];
type TP(P1:T, P1:P) : set of [from T1, from P1];
type M(P1:P) : map <from P1, int>;
type PN(P1:P, T1:T,
        PT1:PT, TP1:TP) : [P1, T1, PT1, TP1];
  
```

Secondly, we describe the specific Petri net instance in WSTSL (Fig. 4). PT1 and TP1 represent the arcs from places to transitions and vice versa, respectively. In tuples, defining arcs, the corresponding transition goes first for the convenience in the description of *Succ* and *Pred* function as it will be seen below.

Fig. 4. Description of the specific Petri net instance in WSTSL

```

var P1:P = {"P1", "P2", "P3", "P4"};
var T1:T = {"T1", "T2"};
var PT1:PT(P1, T1) = [{"T1", "P1"}, {"T2", "P2"},
                     {"T2", "P3"}];
var TP1:TP(T1, P1) = [{"T1", "P2"}, {"T1", "P3"},
                     {"T2", "P1"}, {"T2", "P4"}];
  
```

Then, a well-quasi-ordering should be described (Fig. 5). As it is shown in [3], the element-wise ordering $M \subseteq M'$, when $M(p) \leq M'(p)$ for every place, is a wqo based on the Dickson's lemma [9]. Operator *forall iterator / test* generates a boolean value *true* if the condition *test* is met for each step in *iterator* and a boolean value *false* otherwise.

Fig. 5. Well-quasi-ordering definition in WSTSL

```
func wqo(PN1:PN, s1:M, s2:M)
  return forall p in PN[0] | s1[p] <= s2[p];
end func;
```

As it has been mentioned above in the Algorithms section, Backward Reachability Method requires effective algorithm for computation of *pred-basis*. The algorithm to compute it for Petri Net was suggested in [4]. How it is described in WSTSL is shown in Fig. 6.

Fig. 6. Description of the pred-basis and pred functions in WSTSL

```
func pred(PN1:PN, K:set of M)
  var P1:P = PN1[0];
  var T1:T = PN1[1];
  var PT1:PT(P1,T1) = PN1[2];
  var TP1:TP(T1,P1) = PN1[3];
  var predecessors: set of M(P1) = { };

  for s in K
    for t in T
      if forall tp in TP1[t] | s[tp[1]] - 1 >= 0 then
        s1 = s;
        for pt in PT1[t]
          s1[pt[1]] = s1[pt[1]] + 1;
        end for;
        for tp in TP1[t]
          s1[tp[1]] = s1[tp[1]] - 1;
        end for;
        predecessors = predecessors with s1;
      end if;
    end for;
  end for;
  return predecessors;
end func;

func pb(PN1:PN, K:set of M)
  return min(pred(PN1, I), wqo);
end func;
```

To state the covering problem, the initial state and the state which coverability is required to check should be specified. Afterwards, *backwardanalysis* function should be invoked with appropriate arguments (Fig. 7).

Fig. 7. Description of the initial marking and the marking which coverability it is required to check with Backward Reachability Method invocation

```
var m0:M(P1) = {<"P1",1>, <"P2",0>,
               <"P3",2>, <"P4",1>};
var mc:M(P1) = {<"P1",1>, <"P2",1>,
               <"P3",1>, <"P4",2>};

backwardanalysis(PN1, wqo, pb, m0, mc);
```

The tool provides the user with the output that contains sequence of sets K_i , where $K_0 = \{m_c\}$, $K_{n+1} = pb(K_n)$, their union $\bigcup_{i \in \mathbb{N}} K_i$ and its minimal elements (basis). Finally, it is reported whether the analyzed state (marking) m_c is covered or not (Fig. 8).

Fig. 8. Report of the tool for the backward analysis invocation

```
K0: [{P1=1, P2=1, P3=1, P4=2}]
K1: [{P1=0, P2=2, P3=2, P4=1},
     {P1=2, P2=0, P3=0, P4=2}]
K2: [{P1=1, P2=1, P3=1, P4=1}]
K3: [{P1=0, P2=2, P3=2, P4=0},
     {P1=2, P2=0, P3=0, P4=1}]
K4: [{P1=1, P2=1, P3=1, P4=0}]
K5: [{P1=2, P2=0, P3=0, P4=0}]
Union: [{P1=0, P2=2, P3=2, P4=0},
        {P1=0, P2=2, P3=2, P4=1},
        {P1=1, P2=1, P3=1, P4=0},
        {P1=1, P2=1, P3=1, P4=1},
        {P1=1, P2=1, P3=1, P4=2},
        {P1=2, P2=0, P3=0, P4=0},
        {P1=2, P2=0, P3=0, P4=1},
        {P1=2, P2=0, P3=0, P4=2}]
min(Union): [{P1=0, P2=2, P3=2, P4=0},
             {P1=1, P2=1, P3=1, P4=0},
             {P1=2, P2=0, P3=0, P4=0}]
```

The state $\{P1=1, P2=1, P3=1, P4=2\}$ is not covered

As it has been mentioned above in the Algorithms section, Finite Reachability Tree requires effective algorithm for computation of *Succ*. How it is described in WSTSL is shown in Fig. 9.

Fig. 9. Description of the Succ function in WSTSL

```
func succ(PN1:PN, s:M)
  var P1:P = PN1[0];
  var T1:T = PN1[1];
  var PT1:PT(P1,T1) = PN1[2];
  var TP1:TP(T1,P1) = PN1[3];
  var successors : set of M(P1) = { };

  for t in T
    if forall pt in PT1[t] | s[pt[1]] - 1 >= 0 then
      s1 = s;
      for pt in PT1[t]
        s1[pt[1]] = s1[pt[1]] - 1;
      end for;
      for tp in TP1[t]
        s1[tp[1]] = s1[tp[1]] + 1;
      end for;
      successors = successors with s1;
    end if;
  end for;
  return successors;
end func;
```

To construct Finite Reachability Tree only the initial state should be specified. Afterwards, *forwardanalysis* function should be invoked with appropriate arguments (Fig. 10).

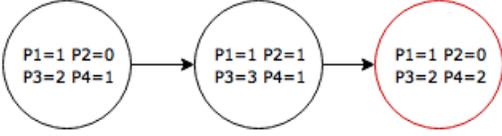
Fig. 10. Description of the initial marking and the Finite Reachability Tree construction invocation in WSTSL

```
var m0:M(P1) = {<"P1",1>, <"P2",0>,
               <"P3",2>, <"P4",1>};

forwardanalysis(PN1, wqo, succ, m0);
```

The tool provides the user with the image which illustrates constructed Finite Reachability Tree (Fig. 11). Nodes are labeled with their states. Dead nodes are red. The node labeled with $\{P1=1, P2=0, P3=2, P4=2\}$ state is dead since $\{P1=1, P2=0, P3=2, P4=2\} \geq \{P1=1, P2=0, P3=2, P4=1\}$ (the latter state is represented by the root which subsumes the dead node labeled by the former state). It demonstrates that the place P4 is not bounded, as the red state has the P4 place large than P4 place in the initial state.

Fig. 11. Constructed finite reachability tree



B. Lossy Channel System

Another model that we considered was Lossy channel system (LCS) which is a subclass of FIFO-channel systems.

Definition 5. FIFO-channel system is a 6-tuple $(S, s_0, A, C, M, \delta)$ where

- S is a finite set of control states;
- $s_0 \in S$ is the initial control state;
- A is a finite set of actions;
- C is a finite set of channels;
- M is a finite set of messages (M^* is a set of finite strings composed of elements from M);
- δ is a finite set of transitions, each of which is represented by one of the following tuples $(s_1, c!m, s_2)$, $(s_1, c?m, s_2)$, (s_1, a, s_2) , where $s_1, s_2 \in S$, $c \in C$, $m \in M$ and $a \in A$ (see below).

Transition $(s_1, c!m, s_2)$ changes the control state from s_1 to s_2 , adding the message m to the end of the channel c . Operation $c!m$ is also known as a send action.

Transition $(s_1, c?m, s_2)$ changes the control state from s_1 to s_2 , removing the message m from the beginning of the channel c . If the channel c is empty or its first element is not m , then this transition cannot occur. Operation $c?m$ is also known as a receive action.

Transition (s_1, c, s_2) changes the control state from s_1 to s_2 and does not change the state of the channels.

In LCS it is also assumed that each message in some channels can be lost at any moment. To model this behavior one more operation $\tau(c, m)$ is introduced.

Transition $(s_1, \tau(c, m), s_2)$ removes the message m from the channel c , and does not change the control state.

For $LCS = (S, s_0, A, C, M, \delta)$ the ordering \leq is defined on the set of global states $\{(s, w) \mid s \in S, w: C \rightarrow M^*\}$ as follows:

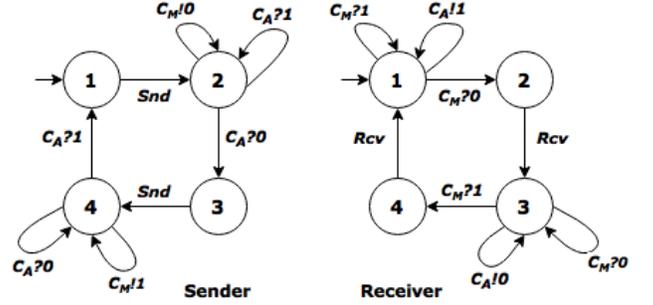
$$(s, w) \leq (s', w') \Leftrightarrow s = s' \wedge w(c) \ll w'(c) \forall c \in C.$$

The ordering \ll is a subword ordering: $u \ll v$ iff u can be obtained by erasing letters from v . It is shown in [6] that this ordering is a wqo; and, LCS systems are WSTS systems.

The concrete model that we considered was Alternating Bit Protocol (ABP). It is represented by Sender and Receiver which communicate via two FIFO-channels c_M and c_A . Sender sends messages to Receiver via c_M , while Receiver sends acknowledgements via c_A . Both channels can lose messages. Messages and acknowledgements contain one-bit sequence

number 0 or 1. Sender continuously sends the same message with the same sequence number, until it receives an acknowledgement from Receiver with the same sequence number. Then, Sender changes (flips) the sequence number and proceeds with sending the next message. Receiver starts by waiting the message with the sequence number 0 (actually, it can initially send acknowledgments with the sequence number 1). When it receives such a message it starts sending acknowledgements with the same sequence number, until it receives the message with the flipped sequence number and so on. The described model is illustrated in terms of Lossy Channel System in Fig. 12.

Fig. 12. Alternating Bit Protocol modelled as a Lossy Channel System



The WSTSL definition of the structure and operators of LCS formalisms is analogously to the Petri Nets formalism WSTSL definition introduced in the Fig. 3-10.

C. Performance

To measure the performance of the implemented Finite Reachability Tree algorithm we applied it to the four different models, which include a model shown in Fig. 2 (Example 1) and the Petri Net models simulating the dining philosophers problem [10] for a number of philosophers equal to 5, 6 and 7. We executed the experiment on the following machine: Intel Core i7, 2.22 GHz, 16 GB RAM running OS X El Capitan (v. 10.11.6). *System.nanoTime()* method was invoked immediately before of the beginning of construction of a FRT and immediately after the end of construction, then the difference was calculated to measure run time for one run. In Table I in the Run time column average results for 20 runs are given in seconds. As well, sizes of the constructed FRTs are given. It can be seen that both run time and size of FRT grow exponentially for the philosophers problem instances.

TABLE I.

	Run time (s)	Size of FRT
Example 1	0.03596	3
Phil5	0.08587	241
Phil6	1.87815	25711
Phil7	5221.64756	88062003

VI. SUMMARY

This paper address a lack of practical results in studies of well-structured transition systems. In order to fill this gap, there was presented one of the possible ways for development of the

system capable to analyze WSTS with two common algorithms: backward reachability method and the Finite Reachability Tree. Well-Structured Transition Systems Language is introduced as a means of describing the user's input, which consists of the description of transition system's structure in general and specific instance's relations and values.

The tool can be used by researchers to investigate the efficiency of the implemented algorithms. It is expected that it is appropriate for conducting experiments on small and medium-sized WSTS. The technology eases the efforts required to check the potential of the WSTS analysis algorithms for practical applications and to make what-if experiments on newly developed formalisms.

The application of the tool is illustrated for the Petri nets and Lossy Channel System formalisms. Also, there were given results of the experiment on Petri nets modeling the dining philosophers problem. The performance analysis of the Finite Reachability Tree applied to this problem demonstrated the expected exponential growth of execution time; and, it indicates the need for further investigations of optimizations (e.g. reduction rules) that can be applied to make the algorithm effectively applicable in practice.

VII. ACKNOWLEDGEMENTS

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

REFERENCES

- [1] J. Burch, E. Clarke, K. McMillan, D. Dill and L. Hwang, "Symbolic model checking: 1020 States and beyond", *Information and Computation*, vol. 98, no. 2, pp. 142-170, 1992.
- [2] A. Finkel, "Well structured transition systems," Univ. Paris-Sud, Orsay, France, Res. Rep. 365, Aug. 1987.
- [3] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!", *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63-92, 2001.
- [4] P. Abdulla, K. Čerāns, B. Jonsson and Y. Tsay, "Algorithmic Analysis of Programs with Well Quasi-ordered Domains", *Information and Computation*, vol. 160, no. 1-2, pp. 109-127, 2000.
- [5] R. Karp and R. Miller, "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147-195, 1969.
- [6] E. Kouzmin and V. Sokolov, *Well-Structured Labeled Transition Systems*, Moscow: Fizmatlit, 2005.
- [7] J. Kruskal, "The theory of well-quasi-ordering: A frequently discovered concept", *Journal of Combinatorial Theory, Series A*, vol. 13, no. 3, pp. 297-305, 1972.
- [8] T. Parr, *The definitive ANTLR 4 reference*, Raleigh, NC and Dallas, TX: The Pragmatic Bookshelf, 2013.
- [9] L. Dickson, "Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors", *American Journal of Mathematics*, vol. 35, no. 4, pp. 413-422, 1913.
- [10] E. Dijkstra, "Hierarchical ordering of sequential processes", *Acta Informatica*, vol. 1, no. 2, pp. 115-138, 1971.
- [11] Akshay S., Genest B., H elou et L., Decidable Classes of Unbounded Petri Nets with Time and Urgency. In: Kordon F., Moldt D. (eds) Application and Theory of Petri Nets and Concurrency. PETRI NETS 2016. *Lecture Notes in Computer Science*, vol 9698. Springer, Cham
- [12] Dworzanski L. W. Consistent Timed Semantics for Nested Petri Nets with Restricted Urgency, in: *Formal Modeling and Analysis of Timed Systems Vol. 9884*, Switzerland : Springer International Publishing, 2016. doi Ch. 1. pp. 3-18.